REMARKS/ARGUMENTS

Claims 1-17 remain in the application. Claim 1 has been amended to correct the certain deficiencies as outlined in the Final Office Action.

The Office Action

1. The Examiner continues to reject claims 1-7 and 17 for containing subject matter that is not described in the specification in such a way as to enable one skilled in the art to make and/or use the invention. Specifically, the Examiner opines that the recited "resource adapters" are not well defined, and that "instructions for creating" are nowhere described in the specification.

In response to applicants' previous arguments on this point, the Examiner states that "the applicant has not provided support or explanation for resource adapters sufficient to allow an understanding of how to use the invention." Without wishing to concede any relevance of Yee et al. (US 6,738,975) as a reference, the Examiner's assertion that it is not clear how a resource adapter provides a uniform interface to access application programming interfaces of resources appears to be inconsistent with her reliance on Yee et al. to establish that it would have an obvious to modify Johnson (US 6,005,925) to include a resource adapter as taught by Yee et al.

In any event, it is respectfully submitted that resource adapters (including their structure and use) were notoriously well-known as of the filing date of this application. As but one example of the numerous well-known technical specifications relating to resource adapters, applicants attach a copy of a draft specification for Java Transaction Service that refers extensively to the operation and design of resource adapters and resource managers and that is completely consistent with Figures 2A and 2B (see, for example, the definition at the bottom of page 4 of the attached document). Many other examples are available that make it clear resource adapters would be well understood by a person of ordinary skill in the art at the time of the invention (*see, e.g.*, U.S. Patents 5,165,031 and 5,613,060).

Accordingly, it is respectfully submitted that the Examiner's rejection based on lack of enablement in connection with the recited "resource adapters" has been traversed.

2. A telephone interview was conducted on August 16, 2007 between the Examiner and applicants' counsel. At that time, the Examiner's rejection of claim 1 based on a lack of enablement with regard to "instructions for creating" was discussed. It was understood that the rejection primarily concerned the lack of consistency between the preamble and the elements of the claim. Accordingly, applicants have amended claim 1 to clarify that the set of program instructions "create agents and adapters for optimizing a bidding process for resources." Applicants respectfully submit that the previous rejection under Section 112, second paragraph is now traversed.

3. The Examiner does not appear to have elaborated on the obviousness rejection set forth in the previous Office Action, other than by responding with a divergent opinion on the correct standard for obviousness. Specifically, in her "Response to Arguments," the Examiner acknowledges that obviousness can only be established by combining or modifying the teachings of the prior art to produce the claimed invention where there is some teaching, suggestion, or motivation to do so found either in the references themselves or in the knowledge generally available to one of ordinary skill in the art, and concludes that in the present case, "the cited references pertain to analogous art i.e. e-commerce, electronic auctions." That said, the mere fact that the prior art could be so modified would not have made the modification obvious unless the prior art suggested the desirability of the modification. *See In re Gordon*, 733 F.2d 900, 221 USPQ 1125 (Fed.Cir. 1984). Further, both the suggestion and the expectation of success must be founded in the prior art, not in applicant's disclosure. *See In re Dow Chemical Co. v. American Cyanamid Co.*, 837 F.2d 469, 5 USPQ2d 1529 (Fed.Cir. 1988). And as noted in *KSR Int'l Co. v. Teleflex, Inc.*, 550 U.S. ___, 127 S.Ct. 1727 (2007), rejections on obviousness grounds cannot be sustained by mere conclusory statements; instead, there must be some articulated reasoning with some rational underpinning to support the legal conclusion of obviousness. That said, it is also true that the prior art references must teach or suggest all of the limitations of the claims. And as noted below, all of the elements of the claims are not taught or suggested by the cited references.

Turning from the debate regarding the legal standard for obviousness to the facts of this application, Johnson is cited for disclosing instructions for creating "a bid manager agent for issuing a call for bids for usage of said resources, receiving said bids and selecting a best bid from among said bids, wherein each of said bids defines a predetermined context." Johnson is further cited for creating "a plurality of bidder agents for issuing said bids according to predetermined bidding policies in response to said call for bids."

The Examiner concedes that Johnson does not specifically disclose "a plurality of resource adapters for providing a uniform interface to access application program interfaces of said resources." Yee et al. is cited for teaching "an adapter in order to transform the data from one application so it can be used by other application [sic]." Examiner's argument is confusing since the claim limitation missing from Johnson is "a plurality of resource adapters for providing a uniform interface to access application program interfaces of said resources," not "...transform the data from one application so it can be used by other application."

In any event, the Examiner concedes that Johnson and Yee et al. do not specifically disclose "issuing said cached bids to said bid manager agent instead of requiring said predetermined bid agents to issue said bid, and a no-caching adapter for receiving from said bid manager agent said call for bids, re-issuing said call for bids to ones of said bidder agents other than said predetermined bidder agents, receiving said bids from said ones of said bidder agents other than said predetermined bidder agents and sending said bids to said bid manager agent,"

The Examiner nonetheless concludes that it would have been obvious "to modify Johnson and Yee to include maintaining a default bid in memory to be used if desired by the bidder in order to provide the bidder with various bidding options according to the bidder's capacity to process the event efficiently" (emphasis added). The Examiner's conclusion is confusing since claim 1 does not recite "maintaining a default bid in memory," but rather "maintaining cached bids for predetermined contexts from predetermined ones of said bidder agents" (emphasis added). The term "context" is defined in the specification at page 4, lines 3-4 as "the set of values that the Bidders need to know in order to calculate their bids accordingly." Thus, if the context of a call is

identical to one that has occurred previously, the bid manager consults the cached bids rather than sending a call for new bids to bidders that have previously submitted bids for that context. (*i.e.*, the "predetermined ones of said bidder agents"). There is absolutely no teaching or suggestion in any of the cited references of applicant's recited "maintaining cached bids for predetermined contexts from predetermined ones of said bidder agents."

The Examiner has also conceded that Johnson, Yee et al. and Baindur do not disclose using a cache to store bids, and cites Kou for teaching a bid cache. The bid cache of Kou is used to store bids from multiple bidders until the closing day of the tender, whereupon the bid requester opens all bid proposals and selects the successful tender. As discussed above in connection with Johnson, Yee et al. and Baindur, there is no teaching or suggestion in Kou of applicant's recited "maintaining cached bids for predetermined contexts from predetermined ones of said bidder agents."

CONCLUSION

For at least the reasons detailed above, it is respectfully submitted all claims remaining in the application (Claims 1-17) are now in condition for allowance.

Respectfully submitted,

FAY SHARPE LLP

8/16/07
Date

John S. Zanghi, Reg. No. 48,843
1100 Superior Avenue, Seventh Floor
Cleveland, OH 44114-2579
216-861-5582

CERTIFICATE OF MAILING OR TRANSMISSION	
I hereby certify that this correspondence (and any item referred to herein as being attached or enclosed) is (are) being ☐ deposited with the United States Postal Service as First Class Mail, addressed to: Mail Stop Amendment, Commissioner for Patents, P.O. Box 1450, Alexandria, VA 22313-1450 on the date indicated below. ☒ transmitted to the USPTO by electronic transmission via EFS-Web on the date indicated below.	
Express Mail Label No.:	Signature: Elaine M. Checovich
Date: 8-20-07	Name: Elaine M. Checovich

N:\PERY\200002\emc0006347V001.docx

Sun Microsystems Inc.

Java Transaction Service (JTS)

This is a draft specification for Java Transaction Service (JTS). JTS specifies the implementation of a transaction manager which supports the JTA specification [1] at the high-level and implements the Java mapping of the OMG Object Transaction Service (OTS) 1.1 Specification at the low-level.

JTS uses the CORBA OTS interfaces for interoperability and portability, which defines a standard mechanism for any implementation that utilizes IIOP (Internet InterORB Protocol) to generate and propagate transaction context between JTS Transaction Managers.

Please send technical comments on this specification to:

jts-spec@eng.sun.com

Version 0.95

Susan Cheung
March 01, 1999

Table of Contents

1 Introduction

This is the Java Transaction Service (JTS) Specification. JTS specifies the implementation of a transaction manager which supports the JTA specification [1] at the high-level and implements the Java mapping of the OMG Object Transaction Service (OTS) 1.1 Specification at the low-level.

JTS uses the CORBA OTS interfaces for interoperability and portability (that is, CosTransactions and CosTSPortability). These interfaces define a standard mechanism for any implementation that utilizes IIOP (Internet InterORB Protocol) to generate and propagate transaction context between JTS Transaction Managers. Note, this also permits the use of other API over the IIOP transport mechanism to be used; for example, RMI over IIOP is allowed.

1.1 Background

Distributed transaction services in Enterprise Java middleware involve five players: the transaction manager, the application server, the resource manager, the application program, and the communication resource manager. Each player contributes to the distributed transaction processing system by implementing different sets of transaction APIs and functionalities.

- A transaction manager provides the services and management functions required to support transaction demarcation, transactional resource management, synchronization, and transaction context propagation.
- An application server (or TP monitor) provides the infrastructure required to support the application run-time environment which includes transaction state management. An example of such an application server is an EJB [5] server.
- A resource manager (through a resource adapter[1]) provides the application access to resources. The resource manager implements a transaction resource interface that is used by the transaction manager to communicate transaction association, transaction completion, and recovery work. An example of such a resource manager is a relational database server.
- A component-based transactional application that operates in a modern application server environment relies on the application server to provide transaction management support through declarative transaction attribute settings—for example, an application developed using the industry standard Enterprise JavaBeans (EJB) component architecture. In addition, other stand-

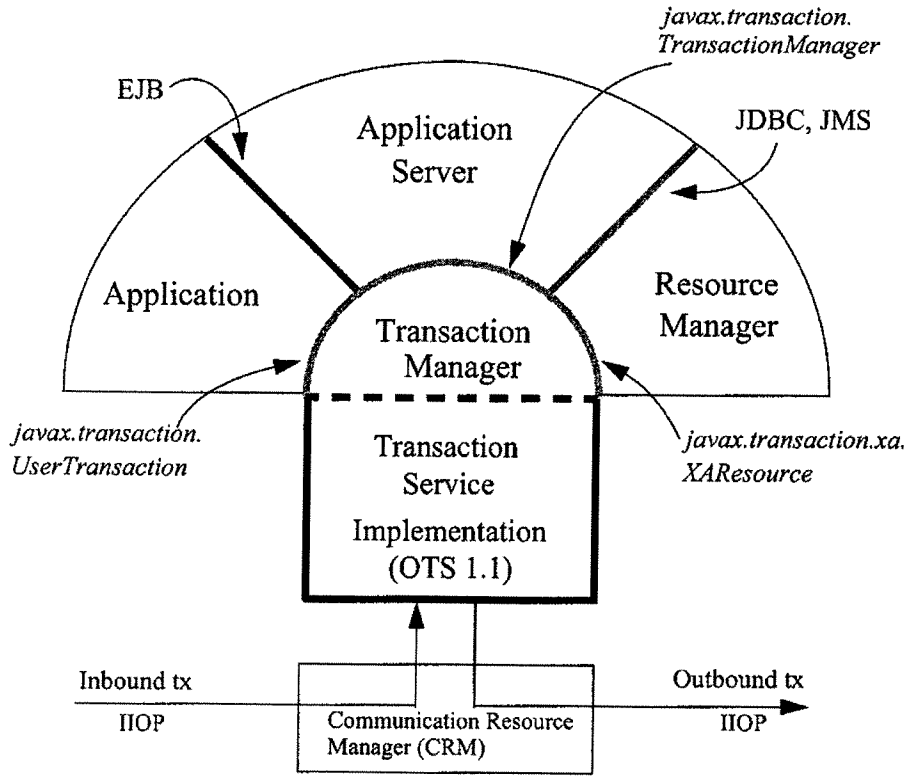
1.A Resource Adapter is a system level software library that is used by an application server or client to connect to a Resource Manager. A Resource Adapter is typically specific to a Resource Manager. It is available as a library and is used within the address space of the client using it. Examples of Resource adapters are: JDBC driver to connect to relational databases, ODMG driver to connect to an object database, JRFC library to connect to SAP R/3 system. A resource adapter may provide additional services besides the connection API.

alone Java client programs may wish to control their transaction boundaries using a high-level interface provided by the application server or the transaction manager.

- A communication resource manager (CRM) supports transaction context propagation and access to the transaction service for incoming and outgoing requests. The JTS document does not specify requirements pertaining to communication. We assume the CRM is present to support transaction propagation as defined in the CORBA OTS and GIOP specifications.

From the transaction manager's perspective, the actual implementation of the transaction services does not need to be exposed; only high-level interfaces need to be defined to allow transaction demarcation, resource enlistment, synchronization, and recovery process to be driven by the users of the transaction services.

The diagram below shows the high-level API exposed from the transaction manager that implements the JTS specification. The dotted-line in the Transaction Manager box illustrates the private interface within the TM to allow the JTA support module to interact with the low-level OTS implementation. Section 2 specifies the Transaction Manager external functionality. Section 3 specifies the Transaction Manager implementation requirements and considerations.



1.2 Target Audience

This document is intended for implementors of Transaction Managers and application servers written in the Java[tm] programming language.

2 Transaction Manager Functionality

This section describes the transaction manager functionality through support of the Java Transaction API (JTA). The implementation of the Java mapping of OMG OTS 1.1 interfaces are not exposed to the clients of the Transaction Manager. The clients of the Transaction Manager are those who use the JTA interfaces to access the Transaction Manager functionality.

The Transaction Manager provides the following services:

- Provides applications and application servers the ability to control the scope and duration of a transaction.
- Allows multiple application components to perform work that is part of a single, atomic transaction.
- Provides the ability to associate global transactions with work performed by transactional resources.
- Coordinates the completion of global transactions across multiple resource managers.
- Supports transaction synchronization.
- Provides the ability to interoperate with other Transaction Manager implementations using the CORBA ORB/TS standard interfaces. (This is transparent to clients of the Transaction Manager.)

2.1 Transaction Model

The Transaction Manager is required to support distributed flat transactions. A flat transaction cannot have a child transaction. Flat transactions are also known as top-level transactions in OTS terminology. A Transaction is started by issuing a request to begin a transaction.

Support for nested transactions is not required.

2.2 Transaction Context

The Transaction Manager maintains the association of a thread's transaction context with a transaction. A thread's transaction context is either *null* or refers to a specific global transaction. The Transaction Manager allows multiple threads to be associated with the same transaction concurrently, in the same JVM or in multiple JVMs.

Transaction context is implicitly transmitted by the implementation of the transaction service at the ORB and wire-protocol level. The transaction context propagation is performed transparent to the Transaction Manager clients (application and application server).

2.3 Transaction Termination

A transaction is terminated by issuing a request to commit or rollback the transaction. Typically, a transaction is terminated by the client originating the transaction. In the

EJB component model environment, the Transaction Manager must allow transactions to be terminated by any thread within the same JVM of the transaction originator.

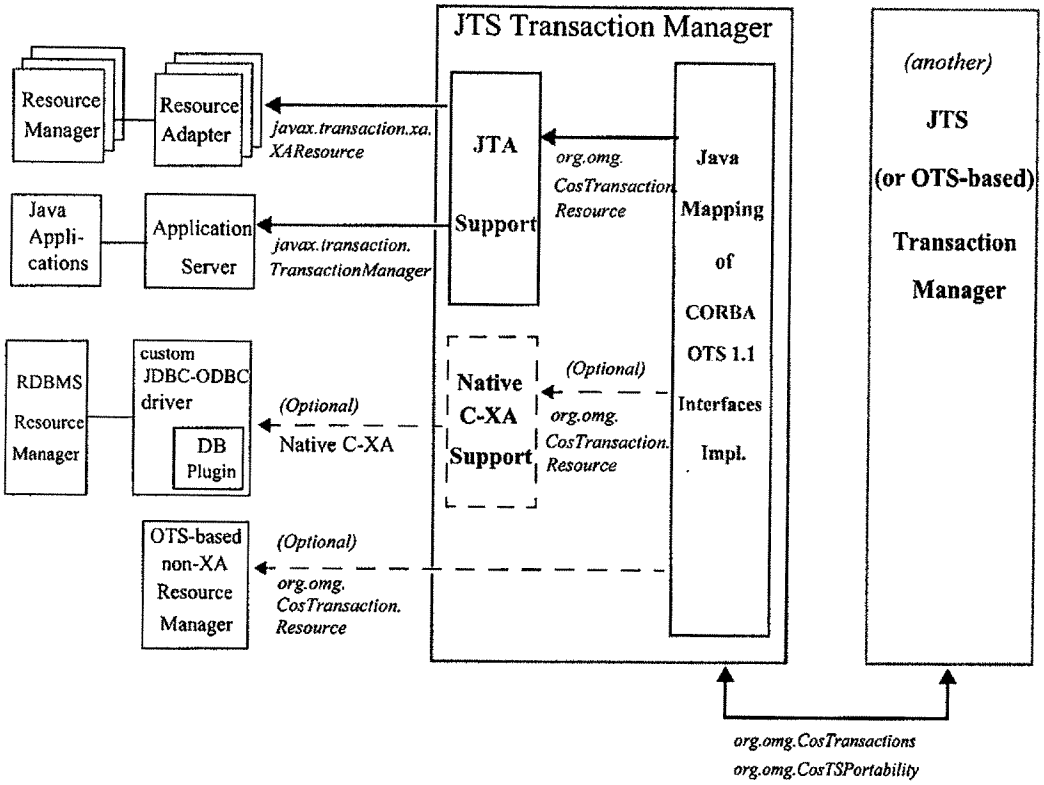
Application components that rely on an application server to manage their transaction states are not allowed to terminate transactions. An application server can force the transaction to be rolled back after the application encounters an unexpected error condition in the form of a Java exception. The Transaction Manager is not required to monitor the failures of the resource managers participating in the transaction.

2.4 Transaction Integrity

The Transaction Manager is required to guarantee data integrity equivalent to that provided by the interfaces which support the X/Open DTP transaction model. The Transaction Manager must guarantee the checked transaction behavior—a transaction cannot be committed until all computations acting on behalf of the transaction have completed.

3 Transaction Manager Implementation

This section describes the implementation choices from a Transaction Manager implementor's view. As shown in the diagram below, the Transaction Manager must implement the JTA interfaces to support the application server and the resource managers. Support for JDBC 1.0 driver and non-JTA aware resource managers is optional. Support for direct CORBA clients, such as recoverable servers and transactional objects, is also optional.



3.1 Support For JTA

The Transaction Manager provides complete support of the Java Transaction API (JTA) Specification [1].

3.1.1 Transaction Demarcation

The Transaction Manager implements the following JTA interfaces to allow application servers and stand-alone Java client applications to control transaction boundary demarcation and perform transaction operations.

- *javax.transaction.TransactionManager*
- *javax.transaction.Transaction*
- *javax.transaction.UserTransaction*

3.1.2 Transaction Synchronization

The Transaction Manager supports transaction synchronization by allowing Synchronization callback objects to be registered by the application server. The Transaction Manager invokes the Synchronization methods before and after transaction completion. Synchronization registration is available via the `javax.transaction.Transaction.registerSynchronization` method.

3.1.3 Transaction and Resource Association

The Transaction Manager supports transactional resource enlistment via the `enlistResource` and `delistResource` methods defined in the `javax.transaction.Transaction` interface.

The Transaction Manager associates resources with transactions and coordinates transaction completion using the `javax.transaction.xa.XAResource` interface as defined in JTA.

3.1.4 Transaction Recovery

The Transaction Manager uses the `recover` and `forget` methods in the `javax.transaction.xa.XAResource` interface to recover transactions that are in prepared or heuristically completed states.

3.2 Java Mapping of CORBA Object Transaction Service (OTS)

The Transaction Manager implements the Java Mapping of the CORBA Object Transaction Service 1.1 Specification [2]. In particular, the Transaction Manager implements the following Java packages: *org.omg.CosTransactions* and *org.omg.CosTSPortability*.

The Transaction Manager is not required to support nested transactions.

The Transaction Manager is not required to expose its OTS implementation to those users who are accessing the Transaction Manager through the `javax.transaction.TransactionManager` interface as defined in JTA.

3.3 Support for Pre-JTA Resource Managers

The Transaction Manager may optionally support pre-JTA resource managers. Specifically, the Transaction Manager may implement a native C-XA support module to provide transaction coordination using the native C-XA procedural interfaces as defined in the X/Open XA Specification [4].

As shown in the previous diagram, to support existing relational database servers that implement the C-XA procedural interface, the Transaction Manager implements a native C-XA support module which uses the `CosTransactions.Resource` interface

to interact with the transaction service module. External to the Transaction Manager, a custom JDBC driver will need to be implemented with a native-XA library built specific to each database server.

3.4 Support for CORBA Applications

The Transaction Manager may optionally support the following CORBA application entities as defined in the Object Transaction Specification: Transactional Client, Transactional Objects, Recoverable Objects, Transactional Servers, and Recoverable Servers. These application entities access the Transaction Manager using the interfaces defined in the *CosTransactions* module as specified in the OTS 1.1 Specification.

3.5 Transaction Managers Interoperability

The Transaction Manager is required to support distributed transactions that involve multiple resource managers in a single ORB execution environment.

If the Transaction Manager implementation supports inter-ORB interoperability, it must implement the implicit transaction context propagation that conforms to the `CosTransactions.PropagationContext` structure; this allows the Transaction Manager to support inter-ORB transaction context propagation as defined by the CORBA OTS 1.1 Specification.

To provide interaction between the ORB and the Transaction Manager, the Transaction Manager is required to

- Implement the `CosTSPortability` module's `Sender` and `Receiver` interfaces as callback objects to allow the ORB to notify the TM whenever a transaction request is sent or received by the ORB.
- Invoke the `TSIdentification` interface methods to pass the `Sender` and `Receiver` objects to the ORB, prior to handling the first transactional request.

How the ORB and the Transaction Manager locate each other's objects is discussed in section 3.6 below. The wire protocol message format for transmitting the transaction context is defined in the CORBA General Inter-ORB Protocol specification.

3.6 ORB Identification

The CORBA OTS 1.1 Specification does not define how the ORB and Transaction Manager identify each other. In order for different ORB instances and the Transaction Manager to interoperate and locate each other, JTS defines a simple `TransactionService` interface to facilitate the identification of the ORB to the Transaction Manager.

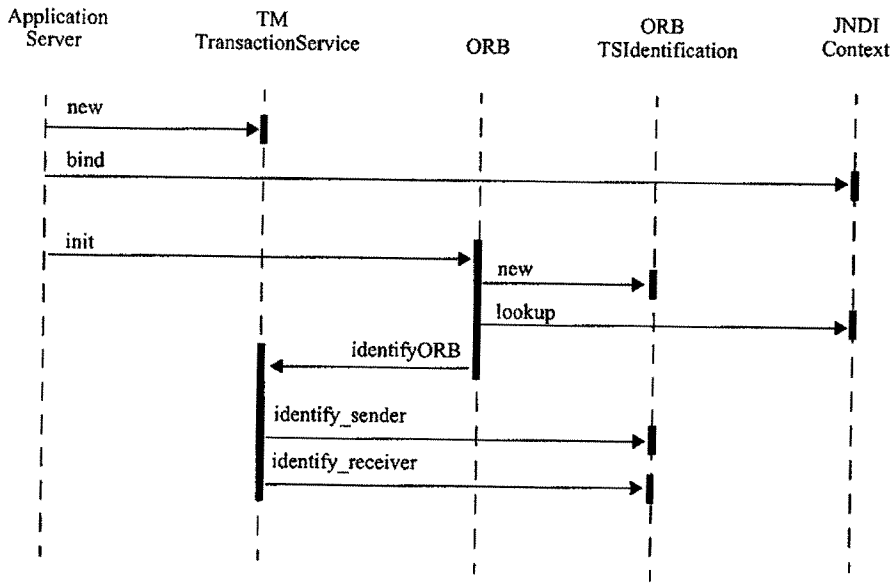
3.6.1 TransactionService Interface

The JTS Transaction Manager implements the `javax.jts.TransactionService` interface to allow an ORB to identify itself to the Transaction Manager.

The ORB calls the `TransactionService.identifyORB` method during its initialization procedure and prior to handling any user request.

Typically, the following operations occur:

1. The application server creates the `TransactionService` object
2. The application server binds the `TransactionService` object to the JNDI naming directory.
3. The application server initializes an ORB instance.
4. The ORB, during its initialization, creates a `TSIdentification` object and uses JNDI to lookup the `TransactionService` object reference.
5. The ORB then invokes the `TransactionService.identifyORB` method and supplies the following three parameters:
   - An ORB object that identifies the ORB instance.
   - A TSIdentification object implemented by the ORB.
   - A properties list for custom configuration information.
6. The Transaction Manager, while executing the `identifyORB` method, invokes the `TSIdentification.identify_sender` and `TSIdentification.identify_receiver` methods to pass the `Sender` and `Receiver` callback objects to the ORB.



Interface TransactionService

```
interface javax.jts.TransactionService {
   public void identifyORB(org.omg.CORBA.ORB orb,
      org.omg.TSIdentification tsi, Properties prop);
}
```

The `javax.jts.TransactionService` interface is implemented by the JTS Transaction Manager to allow the ORB to identify itself to the Transaction Manager and for the Transaction Manager to pass the Sender and Receiver callback objects to the ORB. The Sender and Receiver objects are used by the ORB to deliver the user request's transaction context to the Transaction Manager.

Methods

- **`identifyORB`**

```
public abstract void identifyORB(org.omg.CORBA.ORB orb,
                                 org.omg.CORBA.TSIdentification tsi,
                                 java.util.Properties prop);
)
```

The identifyORB method is called by the ORB as part of its initialization procedure.

Parameters:

`orb`
: The ORB instance

`tsi`
: The TSIdentification object for the TM to identify its Sender and Receiver callback objects.

`prop`
: The properties list for any customed information to the TM.

4 Related Documents

[1] Java Transaction API (JTA) Specification (*http://java.sun.com/products/jta*)

[2] OMG Object Transaction Service (*http://www.omg.org/corba/sectrans.html#trans*)

[3] ORB Portability Submission, OMG document orbos/97-04-14.

[4] X/Open CAE Specification – Distributed Transaction Processing: The XA Specification, X/Open Document No. XO/CAE/91/300 or ISBN 1 872630 24 3

[5] Enterprise JavaBeans™ Specification (*http://java.sun.com/products/ejb*)

[6] JDBC™ 2.0 Standard Extension API Specification (*http://java.sun.com/products/jdbc)*

[7] Java Message Service Specification (*http://java.sun.com/products/jms)*

Appendix A: Change History

A.1 Changes from 0.8 to 0.9

JTS revision 0.9 incorporated the following changes:

- Modified the diagram in Section 3 to include interoperability with another TM.
- Added section 3.6 to specify the `TransactionService` interface which allows the ORB and the TM to locate each other.

A.2 Changes from 0.9 to 0.95

- Added Copyright statement
- Minor editorial changes